Dependency Injection by setter method

1. [Dependency Injection by constructor](https://www.javatpoint.com/spring-tutorial-dependency-injection-by-setter-method)
2. [Injecting primitive and string-based values](https://www.javatpoint.com/spring-tutorial-dependency-injection-by-setter-method#ips)

We can inject the dependency by setter method also. The **<property>** subelement of **<bean>** is used for setter injection. Here we are going to inject

1. primitive and String-based values
2. Dependent object (contained object)
3. Collection values etc.

Injecting primitive and string-based values by setter method

Let's see the simple example to inject primitive and string-based values by setter method. We have created three files here:

* Employee.java
* applicationContext.xml
* Test.java

1. **package** com.javatpoint;
3. **public** **class** Employee {
4. **private** **int** id;
5. **private** String name;
6. **private** String city;
8. **public** **int** getId() {
9. **return** id;
10. }
11. **public** **void** setId(**int** id) {
12. **this**.id = id;
13. }
14. **public** String getName() {
15. **return** name;
16. }
17. **public** **void** setName(String name) {
18. **this**.name = name;
19. }
21. **public** String getCity() {
22. **return** city;
23. }
24. **public** **void** setCity(String city) {
25. **this**.city = city;
26. }
27. **void** display(){
28. System.out.println(id+" "+name+" "+city);
29. }
31. }

**applicationContext.xml**

We are providing the information into the bean by this file. The property element invokes the setter method. The value subelement of property will assign the specified value.

1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans
3. xmlns="http://www.springframework.org/schema/beans"
4. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5. xmlns:p="http://www.springframework.org/schema/p"
6. xsi:schemaLocation="http://www.springframework.org/schema/beans
7. http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
9. <bean id="obj" **class**="com.javatpoint.Employee">
10. <property name="id">
11. <value>20</value>
12. </property>
13. <property name="name">
14. <value>Arun</value>
15. </property>
16. <property name="city">
17. <value>ghaziabad</value>
18. </property>
20. </bean>
22. </beans>

**Test.java**

This class gets the bean from the applicationContext.xml file and calls the display method.

1. **package** com.javatpoint;
3. **import** org.springframework.beans.factory.BeanFactory;
4. **import** org.springframework.beans.factory.xml.XmlBeanFactory;
5. **import** org.springframework.core.io.\*;
7. **public** **class** Test {
8. **public** **static** **void** main(String[] args) {
10. Resource r=**new** ClassPathResource("applicationContext.xml");
11. BeanFactory factory=**new** XmlBeanFactory(r);
13. Employee e=(Employee)factory.getBean("obj");
14. s.display();
16. }
17. }

Setter Injection with Dependent Object Example

1. [Setter Injection with Dependent Object](https://www.javatpoint.com/spring-tutorial-setter-injection-with-dependent-object)

Like Constructor Injection, we can inject the dependency of another bean using setters. In such case, we use **property** element. Here, our scenario is **Employee HAS-A Address**. The Address class object will be termed as the dependent object. Let's see the Address class first:

**Address.java**

This class contains four properties, setters and getters and toString() method.

1. **package** com.javatpoint;
3. **public** **class** Address {
4. **private** String addressLine1,city,state,country;
6. //getters and setters
8. **public** String toString(){
9. **return** addressLine1+" "+city+" "+state+" "+country;
10. }

**Employee.java**

It contains three properties id, name and address(dependent object) , setters and getters with displayInfo() method.

1. **package** com.javatpoint;
3. **public** **class** Employee {
4. **private** **int** id;
5. **private** String name;
6. **private** Address address;
8. //setters and getters
10. **void** displayInfo(){
11. System.out.println(id+" "+name);
12. System.out.println(address);
13. }
14. }

**applicationContext.xml**

The **ref** attribute of **property** elements is used to define the reference of another bean.

1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans
3. xmlns="http://www.springframework.org/schema/beans"
4. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5. xmlns:p="http://www.springframework.org/schema/p"
6. xsi:schemaLocation="http://www.springframework.org/schema/beans
7. http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
9. <bean id="address1" **class**="com.javatpoint.Address">
10. <property name="addressLine1" value="51,Lohianagar"></property>
11. <property name="city" value="Ghaziabad"></property>
12. <property name="state" value="UP"></property>
13. <property name="country" value="India"></property>
14. </bean>
16. <bean id="obj" **class**="com.javatpoint.Employee">
17. <property name="id" value="1"></property>
18. <property name="name" value="Sachin Yadav"></property>
19. <property name="address" ref="address1"></property>
20. </bean>
22. </beans>

**Test.java**

This class gets the bean from the applicationContext.xml file and calls the displayInfo() method.

1. **package** com.javatpoint;
3. **import** org.springframework.beans.factory.BeanFactory;
4. **import** org.springframework.beans.factory.xml.XmlBeanFactory;
5. **import** org.springframework.context.ApplicationContext;
6. **import** org.springframework.context.support.ClassPathXmlApplicationContext;
7. **import** org.springframework.core.io.ClassPathResource;
8. **import** org.springframework.core.io.Resource;
10. **public** **class** Test {
11. **public** **static** **void** main(String[] args) {
12. Resource r=**new** ClassPathResource("applicationContext.xml");
13. BeanFactory factory=**new** XmlBeanFactory(r);
15. Employee e=(Employee)factory.getBean("obj");
16. e.displayInfo();
18. }
19. }

Setter Injection with Collection Example

1. [Setter Injection with Collection](https://www.javatpoint.com/spring-tutorial-setter-injection-with-collection)

We can inject collection values by setter method in spring framework. There can be used three elements inside the **property** element.

It can be:

1. **list**
2. **set**
3. **map**

Each collection can have string based and non-string based values.

In this example, we are taking the example of Forum where **One question can have multiple answers**. There are three pages:

1. **Question.java**
2. **applicationContext.xml**
3. **Test.java**

In this example, we are using list that can have duplicate elements, you may use set that have only unique elements. But, you need to change list to set in the applicationContext.xml file and List to Set in the Question.java file.

**Question.java**

This class contains three properties with setters and getters and displayInfo() method that prints the information. Here, we are using List to contain the multiple answers.

1. **package** com.javatpoint;
2. **import** java.util.Iterator;
3. **import** java.util.List;
5. **public** **class** Question {
6. **private** **int** id;
7. **private** String name;
8. **private** List<String> answers;
10. //setters and getters
12. **public** **void** displayInfo(){
13. System.out.println(id+" "+name);
14. System.out.println("answers are:");
15. Iterator<String> itr=answers.iterator();
16. **while**(itr.hasNext()){
17. System.out.println(itr.next());
18. }
19. }
21. }

**applicationContext.xml**

The list element of constructor-arg is used here to define the list.

1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans
3. xmlns="http://www.springframework.org/schema/beans"
4. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5. xmlns:p="http://www.springframework.org/schema/p"
6. xsi:schemaLocation="http://www.springframework.org/schema/beans
7. http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
9. <bean id="q" **class**="com.javatpoint.Question">
10. <property name="id" value="1"></property>
11. <property name="name" value="What is Java?"></property>
12. <property name="answers">
13. <list>
14. <value>Java is a programming language</value>
15. <value>Java is a platform</value>
16. <value>Java is an Island</value>
17. </list>
18. </property>
19. </bean>
21. </beans>

**Test.java**

This class gets the bean from the applicationContext.xml file and calls the displayInfo method.

1. **package** com.javatpoint;
3. **import** org.springframework.beans.factory.BeanFactory;
4. **import** org.springframework.beans.factory.xml.XmlBeanFactory;
5. **import** org.springframework.core.io.ClassPathResource;
6. **import** org.springframework.core.io.Resource;
8. **public** **class** Test {
9. **public** **static** **void** main(String[] args) {
10. Resource r=**new** ClassPathResource("applicationContext.xml");
11. BeanFactory factory=**new** XmlBeanFactory(r);
13. Question q=(Question)factory.getBean("q");
14. q.displayInfo();
16. }
17. }

Setter Injection with Non-String Collection (having Dependent Object) Example

1. [Setter Injection with Non-String Collection](https://www.javatpoint.com/spring-tutorial-setter-injection-with-non-string-collection)

If we have dependent object in the collection, we can inject these information by using the **ref** element inside the **list**, **set** or **map**. Here, we will use list, set or map element inside the **property** element.

In this example, we are taking the example of Forum where **One question can have multiple answers**. But Answer has its own information such as answerId, answer and postedBy. There are four pages used in this example:

1. **Question.java**
2. **Answer.java**
3. **applicationContext.xml**
4. **Test.java**

In this example, we are using list that can have duplicate elements, you may use set that have only unique elements. But, you need to change list to set in the applicationContext.xml file and List to Set in the Question.java file.

**Question.java**

This class contains three properties, two constructors and displayInfo() method that prints the information. Here, we are using List to contain the multiple answers.

1. **package** com.javatpoint;
3. **import** java.util.Iterator;
4. **import** java.util.List;
6. **public** **class** Question {
7. **private** **int** id;
8. **private** String name;
9. **private** List<Answer> answers;
11. //setters and getters
13. **public** **void** displayInfo(){
14. System.out.println(id+" "+name);
15. System.out.println("answers are:");
16. Iterator<Answer> itr=answers.iterator();
17. **while**(itr.hasNext()){
18. System.out.println(itr.next());
19. }
20. }
22. }

**Answer.java**

This class has three properties id, name and by with constructor and toString() method.

1. **package** com.javatpoint;
3. **public** **class** Answer {
4. **private** **int** id;
5. **private** String name;
6. **private** String by;
8. //setters and getters
10. **public** String toString(){
11. **return** id+" "+name+" "+by;
12. }
13. }

**applicationContext.xml**

The **ref** element is used to define the reference of another bean. Here, we are using **bean** attribute of **ref** element to specify the reference of another bean.

1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans
3. xmlns="http://www.springframework.org/schema/beans"
4. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5. xmlns:p="http://www.springframework.org/schema/p"
6. xsi:schemaLocation="http://www.springframework.org/schema/beans
7. http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
9. <bean id="answer1" **class**="com.javatpoint.Answer">
10. <property name="id" value="1"></property>
11. <property name="name" value="Java is a programming language"></property>
12. <property name="by" value="Ravi Malik"></property>
13. </bean>
14. <bean id="answer2" **class**="com.javatpoint.Answer">
15. <property name="id" value="2"></property>
16. <property name="name" value="Java is a platform"></property>
17. <property name="by" value="Sachin"></property>
18. </bean>
20. <bean id="q" **class**="com.javatpoint.Question">
21. <property name="id" value="1"></property>
22. <property name="name" value="What is Java?"></property>
23. <property name="answers">
24. <list>
25. <ref bean="answer1"/>
26. <ref bean="answer2"/>
27. </list>
28. </property>
29. </bean>
31. </beans>

**Test.java**

This class gets the bean from the applicationContext.xml file and calls the displayInfo method.

1. **package** com.javatpoint;
3. **import** org.springframework.beans.factory.BeanFactory;
4. **import** org.springframework.beans.factory.xml.XmlBeanFactory;
5. **import** org.springframework.core.io.ClassPathResource;
6. **import** org.springframework.core.io.Resource;
8. **public** **class** Test {
9. **public** **static** **void** main(String[] args) {
10. Resource r=**new** ClassPathResource("applicationContext.xml");
11. BeanFactory factory=**new** XmlBeanFactory(r);
13. Question q=(Question)factory.getBean("q");
14. q.displayInfo();
16. }
17. }

Setter Injection with Map Example

1. [Setter Injection with Map Example](https://www.javatpoint.com/spring-tutorial-setter-injection-with-map)

In this example, we are using **map** as the answer for a question that have answer as the key and username as the value. Here, we are using key and value pair both as a string.

Like previous examples, it is the example of forum where **one question can have multiple answers**.

**Question.java**

This class contains three properties, getters & setters and displayInfo() method to display the information.

1. **package** com.javatpoint;
2. **import** java.util.Iterator;
3. **import** java.util.Map;
4. **import** java.util.Set;
5. **import** java.util.Map.Entry;
7. **public** **class** Question {
8. **private** **int** id;
9. **private** String name;
10. **private** Map<String,String> answers;
12. //getters and setters
14. **public** **void** displayInfo(){
15. System.out.println("question id:"+id);
16. System.out.println("question name:"+name);
17. System.out.println("Answers....");
18. Set<Entry<String, String>> set=answers.entrySet();
19. Iterator<Entry<String, String>> itr=set.iterator();
20. **while**(itr.hasNext()){
21. Entry<String,String> entry=itr.next();
22. System.out.println("Answer:"+entry.getKey()+" Posted By:"+entry.getValue());
23. }
24. }
25. }

**applicationContext.xml**

The **entry** attribute of **map** is used to define the key and value information.

1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans
3. xmlns="http://www.springframework.org/schema/beans"
4. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5. xmlns:p="http://www.springframework.org/schema/p"
6. xsi:schemaLocation="http://www.springframework.org/schema/beans
7. http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
9. <bean id="q" **class**="com.javatpoint.Question">
10. <property name="id" value="1"></property>
11. <property name="name" value="What is Java?"></property>
12. <property name="answers">
13. <map>
14. <entry key="Java is a programming language"  value="Sonoo Jaiswal"></entry>
15. <entry key="Java is a Platform" value="Sachin Yadav"></entry>
16. </map>
17. </property>
18. </bean>
20. </beans>

**Test.java**

This class gets the bean from the applicationContext.xml file and calls the displayInfo() method.

1. **package** com.javatpoint;
3. **import** org.springframework.beans.factory.BeanFactory;
4. **import** org.springframework.beans.factory.xml.XmlBeanFactory;
5. **import** org.springframework.core.io.ClassPathResource;
6. **import** org.springframework.core.io.Resource;
8. **public** **class** Test {
9. **public** **static** **void** main(String[] args) {
10. Resource r=**new** ClassPathResource("applicationContext.xml");
11. BeanFactory factory=**new** XmlBeanFactory(r);
13. Question q=(Question)factory.getBean("q");
14. q.displayInfo();
16. }
17. }

Setter Injection with Non-String Map (having dependent Object) Example

1. [Setter Injection with Non-String Map](https://www.javatpoint.com/spring-tutorial-setter-injection-with-non-string-map)

In this example, we are using **map** as the answer that have Answer and User. Here, we are using key and value pair both as an object. Answer has its own information such as answerId, answer and postedDate, User has its own information such as userId, username, emailId.

Like previous examples, it is the example of forum where **one question can have multiple answers**.

**Question.java**

This class contains three properties, getters & setters and displayInfo() method to display the information.

1. **package** com.javatpoint;
2. **import** java.util.Iterator;
3. **import** java.util.Map;
4. **import** java.util.Set;
5. **import** java.util.Map.Entry;
7. **public** **class** Question {
8. **private** **int** id;
9. **private** String name;
10. **private** Map<Answer,User> answers;
12. //getters and setters

15. **public** **void** displayInfo(){
16. System.out.println("question id:"+id);
17. System.out.println("question name:"+name);
18. System.out.println("Answers....");
19. Set<Entry<Answer, User>> set=answers.entrySet();
20. Iterator<Entry<Answer, User>> itr=set.iterator();
21. **while**(itr.hasNext()){
22. Entry<Answer, User> entry=itr.next();
23. Answer ans=entry.getKey();
24. User user=entry.getValue();
25. System.out.println("Answer Information:");
26. System.out.println(ans);
27. System.out.println("Posted By:");
28. System.out.println(user);
29. }
30. }
31. }

**Answer.java**

1. **package** com.javatpoint;
3. **import** java.util.Date;
5. **public** **class** Answer {
6. **private** **int** id;
7. **private** String answer;
8. **private** Date postedDate;
9. **public** Answer() {}
10. **public** Answer(**int** id, String answer, Date postedDate) {
11. **super**();
12. **this**.id = id;
13. **this**.answer = answer;
14. **this**.postedDate = postedDate;
15. }
17. **public** String toString(){
18. **return** "Id:"+id+" Answer:"+answer+" Posted Date:"+postedDate;
19. }
20. }

**User.java**

1. **package** com.javatpoint;
3. **public** **class** User {
4. **private** **int** id;
5. **private** String name,email;
6. **public** User() {}
7. **public** User(**int** id, String name, String email) {
8. **super**();
9. **this**.id = id;
10. **this**.name = name;
11. **this**.email = email;
12. }
14. **public** String toString(){
15. **return** "Id:"+id+" Name:"+name+" Email Id:"+email;
16. }
17. }

**applicationContext.xml**

The **key-ref** and **value-ref** attributes of entry **element** is used to define the reference of bean in the map.

1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans
3. xmlns="http://www.springframework.org/schema/beans"
4. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5. xmlns:p="http://www.springframework.org/schema/p"
6. xsi:schemaLocation="http://www.springframework.org/schema/beans
7. http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
9. <bean id="answer1" **class**="com.javatpoint.Answer">
10. <property name="id" value="1"></property>
11. <property name="answer" value="Java is a Programming Language"></property>
12. <property name="postedDate" value="12/12/2001"></property>
13. </bean>
14. <bean id="answer2" **class**="com.javatpoint.Answer">
15. <property name="id" value="2"></property>
16. <property name="answer" value="Java is a Platform"></property>
17. <property name="postedDate" value="12/12/2003"></property>
18. </bean>
20. <bean id="user1" **class**="com.javatpoint.User">
21. <property name="id" value="1"></property>
22. <property name="name" value="Arun Kumar"></property>
23. <property name="email" value="arun@gmail.com"></property>
24. </bean>
25. <bean id="user2" **class**="com.javatpoint.User">
26. <property name="id" value="2"></property>
27. <property name="name" value="Varun Kumar"></property>
28. <property name="email" value="Varun@gmail.com"></property>
29. </bean>
31. <bean id="q" **class**="com.javatpoint.Question">
32. <property name="id" value="1"></property>
33. <property name="name" value="What is Java?"></property>
34. <property name="answers">
35. <map>
36. <entry key-ref="answer1" value-ref="user1"></entry>
37. <entry key-ref="answer2" value-ref="user2"></entry>
38. </map>
39. </property>
40. </bean>
42. </beans>

**Test.java**

This class gets the bean from the applicationContext.xml file and calls the displayInfo() method to display the information.

1. **package** com.javatpoint;
3. **import** org.springframework.beans.factory.BeanFactory;
4. **import** org.springframework.beans.factory.xml.XmlBeanFactory;
5. **import** org.springframework.core.io.ClassPathResource;
6. **import** org.springframework.core.io.Resource;
8. **public** **class** Test {
9. **public** **static** **void** main(String[] args) {
10. Resource r=**new** ClassPathResource("applicationContext.xml");
11. BeanFactory factory=**new** XmlBeanFactory(r);
13. Question q=(Question)factory.getBean("q");
14. q.displayInfo();
16. }
17. }